An MDE Approach for Domain based Architectural Components Modelling.

Rahma Bouaziz

An MDE Approach for Domain based Architectural Components Modelling

Rahma Bouaziz
IRIT, University of Toulouse
118 Route de Narbonne, 31062 Toulouse, France
rbouaziz@irit.fr

ABSTRACT—Component Based Software Engineering (CBSE) is a popular and widely adopted software engineering paradigm that has proven his usefulness and success to increase reusability and efficiency in various application domains. In this paper, we propose a common metamodel of a component to support all the requirements of CBSE taking into account the specificities of each domain. The resulting modeling framework serves primarily to capture the basic concepts of concerns related to component systems development based on the clear separation between the development process, interactions and the domain knowledge. As a proof of concept, we are evaluating the feasibility of our approach through the CCM component model applied to an use case for building systems having real-time requirements.

Keywords—Component Based Software engineering, Meta-modeling, Component Models, Model Driven Engineering.

I. INTRODUCTION

Component-Based Software Engineering (CBSE) [1] has emerged as a promising key technology for developing and maintaining complex systems. CBSE focuses on building large software systems by integrating previously existing software components. The system is constructed by the composition and the connection of these components. It is a good solution to optimize time and cost of software design while still guaranteeing the quality of the software [2] [3].

Various component models have been proposed to deal with system complexity in industrial and academic domains. Variety of those model's applications in constructing systems has proved their usefulness and success. Among these approaches, we can find general-purpose software component models such as Enterprise Java Beans (EJB) [6], CORBA Component Model (CCM) [9] or, Fractal [10] which are well-established for CBSE in generic problem domains. They provide basic mechanisms for the specification and the composition of components. On the other hand, to address a specific domain challenge, specific component models like AUTOSAR [11], BIP [8], and KOALA [7] are proposed to deal with specialized domains like distributed, embedded or real time systems [21].

Our first objective is to combine these two approaches – generic and specific component models – in order to propose a metamodel that overcome some of drawbacks and take advantage of each approach. In other terms, we propose a common representation of generic and specific component models taking into account domain specific concerns at the design level.

Model-Driven Engineering (MDE) [5] is also another approach emerging in system development. The use of models has become a major paradigm in software engineering. Its use represents a significant advance in terms of level of abstraction, continuity, generality, and scalability. MDE is a form of generative engineering, in which all or a part of an application is generated from models. It looks promising since it offers tools to deal with the development of complex systems improving their quality and reducing their development cycles. The development is based on metamodeling, development process and execution platforms.

In this paper, we deal with these two technologies - CBSE and MDE - to propose a model-based component framework to get a common representation of component for several domains. The main motivation of this work is that the reuse of knowledge and expertise at high level is fundamental to guarantee quality systems. The architecture of the proposed component modeling framework is composed of: (i) a Component Metamodel inspired from the popular CBSE principles and (ii) two model levels: domain independent and domain specific.Inspired by the MDE methodology in which software is developed by constructing high level models, we propose a generic component metamodel - to capture generic concepts of CBSE approach – and then separate domain independent aspects of component model from those that are domain specific.

After this introduction, the remainder of this article is organized as follows: in Section II, we present an overview and related works of component approach and domain approach by reviewing the basic component model concepts. Section III introduces the structure of the proposed solution. After that, in Section IV, the proposed metamodel is presented. An illustration of the approach is presented in Section V through CCM component model. Finally, the main achievements and future intentions are summarized in the concluding Section VI.

II. RELATED WORKS

In this section we will focus on component approaches presenting a short overview of the main general- purpose component models and models targeting special domains.

A. Component Approaches

In last decades, several definitions of component have been proposed, in this work we will adopt one of the most commonly definition proposed by Szyperski in 2002 [12], “A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to third-party composition.”

In this sub-section, we present a review of the component models - industrial as well as academic -studied in this work. Among works that discuss a general-purpose component-based systems-building technology, we take as component technologies reference CCM, Fractal, Sofa 2.0, EJB and UML 2.0 [13]. Abstract models defined by these technologies are used to define an application in the form of components that provide a number of services (interfaces) and explaining their dependencies with their environment.

Enterprise Java Beans (EJB) [6] developed by Sun MicroSystems envision the construction of object-oriented and distributed business applications. It is a software component model for developing and deploying computing applications that are scalable, transactional, and multi-user secure. The model simplifies the development of middleware by providing server support for services such as transactions, security, database connectivity, and component customization [24]. It is one of the most used technologies in component-based system implementation.

Each component, in OMG CORBA Component Model (CCM) [9] is defined by its attributes and ports. Attributes are used for configuration and ports represent the interfaces through which customers and other environmental elements of an application can interact. This model component supports four types of ports – (1) a facet (provided interface), (2) a receptacle (required interface), (3) an event source (emits events) and an event sink (receives events) - and (4) attributes.

Fractal [10] component model is defined by France Telecom R&D and INRIA. Fractal is a general purpose component model. It uses a hierarchical component model without connectors. Fractal component is formed of content and a membrane. The content consists of a finite number of components called sub-components. The membrane consists of functional interfaces corresponding to the functionality provided or required and control interfaces corresponding to non-functional aspects. The interfaces of a membrane may be internal or external. The external interfaces are accessible from outside the component, while the internal interfaces are only accessible to its sub-components. Binding in the Fractal model is what allows Fractal components to communicate. Composition is possible through the notion of composite component-a component that contains sub-components-. However, a primitive component contains no sub-component.

In the SOFA 2.0 components model [17], a component can be primitive (does not contain any sub-component) or composite (contains sub-components). All features are defined in the primitive components. A component is described by its frame and architecture. The frame is a view "black box" component. It defines the interfaces provided and required, and eventually declares properties for component's parameterization. The architecture reflects a view "gray box" component. It defines the first level of nesting in a hierarchy of components. The components are connected by connectors. In the SOFA model, the connectors are entities such as components. Each connector is described in the same way that a component, i.e. by a frame and architecture.

On the other hand, UML [13] provides a unified notation for modeling and a standard supported by several software engineering tools. In its 2.0 specification [11] (adopted by OMG), UML standard provides the necessary elements to describe both abstract architectures and implementations based components. However, UML is a modeling language for general purpose. Therefore, there is a multitude of concepts in the meta-model which make it quite complex. The component model of UML 2.0 [11] offers various concepts to describe the components: Component, Interface, Port and Connector.

A synthesis of the architectural elements supported by the models discussed above is illustrated in Fig.1, i.e., for each approach we give principle concepts.

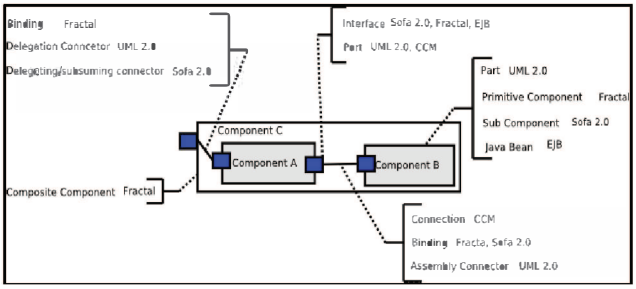

Fig. 1. A synthesis of the concepts supported by component models (CCM, SOFA 2.0, Fractal, and EJB) and UML 2.0

In the other hand, we can find some works that try to propose a generic component metamodel. OpenCom, [14] for instance, propose a generic component-based approach for the construction of systems software. Authors assume that a general-purpose systems-building technology should provide only generic and fundamental functionality that is independent of the specialist needs of any particular target domain. They try to address the following requirements: (1) Target domain independence and (2) Deployment environment independence.

A generic component model (GCM) is proposed in the MARTE [15] (Modeling and Analysis of real-time and embedded systems) profile specification proposed by the OMG (Object Management Group). It aims to add capabilities to UML for the model driven development of real-time and embedded systems.

A French national project, Flex-eWare [16], targets the building of a common platform for component based design of embedded systems. FCM is a meta-model inspired from UML, Fractal and CCM. the aim of this FCM metamodel is to unify the underlying component models concepts. A first key objective of Flex-eWare is to federate the two Lightweight CCM and Fractal models.

One of the major limitations of presented Generic Component Models cited above like OpenCom and FCM is that these models do not take into account domain specific concerns at the design level like real time or distributed requirements. Therefore, we propose a Generic Component Metamodel aiming to reflect generic and specific domain concepts.

B. Domain Approaches

There are various definitions for domain in the context of component engineering. In this work, we adopted the definition proposed by Crnkovic in 2011 [5]. In which a domain is presented as an area in which application and business domains component models are used or supposed to be used. It indicates the specialization, or the generality of component models."

Authors propose a classification of the component models in three major classes: (1) A general-purpose component models, (2) A specialized component models and (3) A generative component models. In this work we will focus in both first classes.

In the previous sub-section, we considered general purpose component models like CCM and EJB. In this sub-section, we will present some specialized component models like AUTOSAR [11], BIP [8], BlueArX, COMDES II, and KOALA [7]. The main characteristic of such models is the integration of requirements from the application domain into the component model.

Recently, various proposals dealing with the adaptation of CBSE to several domains have been proposed. In the industrial field, and to integrate real-time requirements, some companies have developed their own solutions, adapted to their corresponding domains (e.g. Railways, Automotive, etc). Volvo is one of these industrial companies which use Koala, developed by Philips, and Rubus proposed by Lundbäck *et al.* [20].

Koala [7] is a component model for embedded devices developed by Philips. Koala is inspired from the COM [32] model and Darwin component models [28]. It's goals are to manage increasing complexity and diversity of software used mostly in consumer electronics. Koala basic elements are components defining set of provided and required interfaces. The component model supports hierarchical components (called compound components).

In the automotive domain, the AUTOSAR (AUTomotive Open System AR-chitecture) consortium [11] is the first large-scaled initiative to gather manufacturers, suppliers and tool developers from the automotive field to establish open and standardized software architecture for the automotive domain enabling component-based software design modeling. Through this common standard, the vision of AUTOSAR is to facilitate the exchange of solutions (including software components) between different vehicle platforms and subsystem manufacturers as well as between vehicle product lines. In that sense, AUTOSAR targets the upper part of the granularity scale of the proposed conceptual component model.

General purpose component models - presented in the previous sub-section - can be adopted for particular domains as embedded, distributed systems or real time systems- by addition of new features and functionalities.

Sofa HI [19] is an extension of SOFA 2.0 component model targeted at high-integrity real-time embedded systems. However, key features of SOFA HI remain compatible with SOFA 2.0. The goal of SOFA HI is to bring the knowledge of component systems gained from SOFA and SOFA 2.0 development into the real-time environment to speed up the development and lower the costs of high-integrity systems.

The ARINC Component Model (ACM) [22] is a component model for Hard Real Time Systems. It is an extension of the CORBA Component Model (CCM) for real time. ACM is steps towards a hard real-time component model based on CCM that provides the essential component abstraction and ARINC [18] to provide the platform (API).

SaveComp (SaveComp component model) [23] is a domain specific component model targeting safety-critical hard real-time embedded systems, developed at Mälardalen University. It being developed in the project SAVE (Component Based Design of Safety Critical Vehicular Systems). It is designed for embedded control applications in the automotive domain with the main objective of providing predictable vehicular systems. SaveCCM is a simple model that constrains the flexibility of the system in order to improve the analysability of the dependability and of the real-time properties [5].

The Lightweight CORBA Component Model [25] (LwCCM) is an OMG specification standardizes the development, configuration, and deployment of component-based applications. LwCCM uses CORBA's distributed object computing model as its underlying architecture, so applications are not tied to any particular language or platform for their implementations. Components in LwCCM are the implementation entities that export a set of interfaces usable by conventional middleware clients as well as other components. Components can also express their intent to collaborate with other components by defining ports- facets, receptacles and event sources/sinks – [26].

Based on the state of the art presented above, we can say that the difficulty became to define a sufficiently rich and generic metamodel providing abstraction and generality of chosen component models integrating domain independent and domain specific levels.

III. OVERVIEW OF THE APPROACH

Taking into account related works presented bellow, the key idea presented in this work is to propose a common representation to target several domains of systems applications (e.g. distributed, embedded, real time, etc.). This representation allows us to work at a higher

abstraction level, which may significantly reduce the cost of system engineering. Our goal is to define the component based systems as easily and quickly as possible. To do this, the modeling framework must include simple and known abstractions by the software developer. Our proposed architecture is based on models, which specify different levels of abstraction, helping developers to manage the inherent complexity of applications and facilitating the communication between the different contributors of software development. Fig. 2 presents an overview of the proposed architecture based on different models:

Generic Component Meta-Model (GeCM). In This level -Metamodel level (M2) – we present a generic component metamodel that represents as its name suggests the abstract concepts of component based approach proposed by a large set of component models to describe software architectures. It provides the basic modeling elements for component based system: Component, Connector, Interface, Ports, etc. These elements are the basis for instantiating different component model. This metamodel is detailed in section 4. GeCM will be used to describe both domain-independent and domain specific models.

Domain Independent Component Model (DICM). This model is an instance of the GeCM. This level is intended to generically represent component independently from the application domain. Hence, we will focus on the representation of generic component concepts, interaction and connection between component conforming to the component metamodel (GeCM). Here we focus on component models presented in the sub-section 2.1. A DICM model is conform to the GeCM Meta-model. Therefore each element of DICM is associated with an element of GeCM.

Domain Specific Component Model (DSCM). This model corresponds to a refinement of the DICM model. It combines component specification presented in DICM and the domain specification such real-time constraints. This model is, at the same time, an instantiation of the Generic Component model (GeCM) and a refinement of DICM. In this stage the DICM is tailored to a particular application domain e.g. real time embedded or distributed system. For example we can build from the Fractal model (DICM) several specific component models (DSCM) like Fractal distributed, Fractal real-time or Fractal embedded component models.

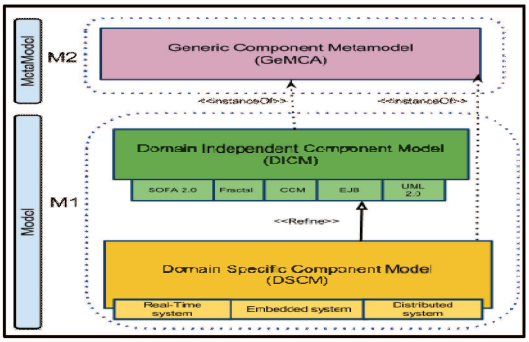

Fig. 2. Overview of the proposed Architecture

IV. GeCM: A Generic Component Metamodel

Based on our study of selected component models as shown in section II, we propose GeCM. In this work, we try to merge 1/ The component technologies: Enterprise JavaBeans (EJB) from Sun Microsystem, 2/ The CCM (CORBA Component Model) of the OMG, 3/ The Fractal hierarchical component model of the ObjectWeb consortium, 4/ The part of the UML 2.0 metamodel concerned with the components and 5/ The SOFA 2.0 components model. GeCM provides a component model state-of-the-art offering most features available in the domain of component-oriented software architectures.

GeCM describes concepts and their relations needed to represent component systems. The GeCM is based on CBSE principles, containing the basic entities *Component*, *Interfaces* and *Connector*. Fig. 3 presents the proposed meta-model. It defines the abstractions of component based concepts. These components will be used to describe both Domain-Independent (DI) and Domain-Specific (DS) models i.e., the originality is the use of the domain independent model and the domain specific model which are reflected in the metamodel level. In the proposed metamodel, a system is described by a set of components that represents the units of computation and data. The communication mode and coordination between these components is encapsulated in connectors. GeCM metamodel separates the computation concept represented by components and the interaction concept presented by connectors.

As shown in Fig.3 the generic metamodel defines the abstractions of component based concepts and allows the design of generic component based applications.

In GeCM component concepts are defined as follows:

ArchitecturalElement. Is an abstract class. This class is associated to itself, using a reflexive association “Refine”. That’s mean that a class’s instance refine another instance of the class. In other word, a domain specific architectural element “Refine” a domain independent architectural element.

Component. Is a first class architectural element in a system. It can be primitive or composite (derived from existing components) and possesses one or more *attributes*. Component communicates with its environment through an interface with multiple ports. Those ports can be required or provided one or more point of interaction. It can be independently deployed and composed. Primitive component as well as Composite Component can be domain independent or domain specific. We note domain independent component as DI_Composite/primitive Component. A DS_Component Refine a DI_Component.

Interface. Component interacts with its environment with Interfaces which are composed of operations. So, larger pieces of a systems functionality may be assembled by reusing components as parts in an encompassing component or assembly of components, and wiring together

their required and provided interfaces. More precisely, a component owns provided and required interfaces. A provided interface is implemented by the component and highlights the services exposed to the environment. A required interface corresponds to services needed by the component to work properly. There are two types of interface: the operation and the port.

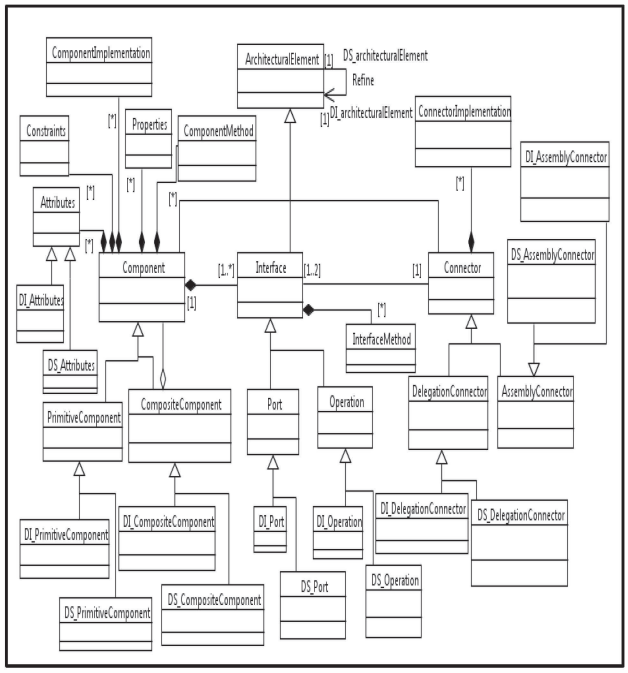

Fig. 3. GeCM: A Generic Component Metamodel

Port. Is a point of interaction that expresses a potential collaboration between two components. A port can be domain independent or domain specific. A DS_Port refine a DI_Port.

Operation. The other type of interface is operation. We define domain independent operation which is refined by a domain specific operation.

Connector. Is a first class architectural element that reflects the specific features of interactions among components in a system. It allows the interaction of the component with it's environment. They provide the link between architectural elements. We distinguish two types of connectors:

Assembly connector. Is used for construction links (to connect interfaces of different types). A domain specific assembly connector refines domain independent assembly connector.

Delegation connector. Is used to link a port of the composite component to a port of a component located within a composite component (to connect interfaces of same type). A domain independent delegation connector can be refined by a domain specific delegation connector.

Constraints. Define certain rules that should be respected in order to ensure adherence of architectural element's use.

Properties. Represent additional information about component.

In addition to these classes and associations, relations can be more precisely defined by adding constraints. These constraints are typically expressed in OCL (Object Constraint Language) language [27]. The set of instances of the proposed GeCM meta-model can be restricted by additional OCL constraints. A set of constraints are also defined using the Object Constraint Language (OCL). Because a lack of space we will not present OCL constraints in this paper.

V. Illustration of the approach

To illustrate the approach, CORBA Component Model (CCM) is presented as a Domain Independent Component Model, (DICM) and as an instance of the proposed component metamodel (GeCM). This component model can be specialized for several domains as real Time, embedded or distributed domain.

As shown in Fig. 2, at DSCM level (domain specific level) one or several models can be built according to the architecture described at the above level (DICM). Each architectural element of the domain specific level is a refinement of an element of the DICM. For example we can build from the CCM model (DICM), the CCM real-time component model as a domain specific model. We present in this illustration an example of the ARINC Component Model (ACM) which represents the refinement of the CCM component model to the real-time domain.

C. CORBA Component Model

In this section, we illustrate our proposal using CORBA Component Model (CCM) as DICM and the ARINC Component Model (ACM) as DSCM.

1) CCM as an example of DICM

This level is intended to generically represent component independently from the application domain. Among existing component technologies, we studied the OMG CORBA Component Model (CCM). Figure 4 illustrates these concepts and shows an example of how CCM is instantiated from GeCM. As presented in Fig 4, we can conclude that each CCM notation is an instance of GeCM notations. For example a component is an instance of DI_PrimitiveComponent, also, port and event are instances of DI_Port. However Connection is an instance of DI_AssemblyConnector. In CORBA Component Model specification, a component can be derived from existing components (composite component). A CORBA component defines a set of ports. A component may declare a set of attributes. Attributes can be used to configure an instance of a CORBA component as a name, a type and a value. An interface is characterized by a set of operations. Connector links are provided to a used port through interfaces. External interfaces are a required or provided interfaces (facets or receptacles) used in component assembly. Facets are provided interfaces that define the services provided by the component to other components.

Receptacles correspond to the interfaces required by a component to function in a given environment.

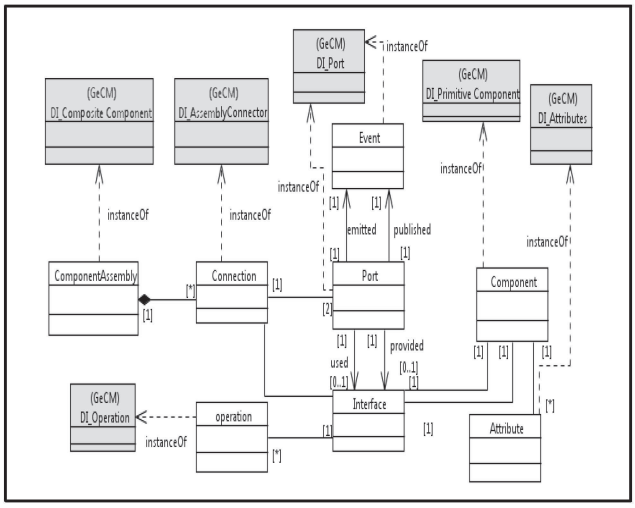

Fig. 4. . Instantiating the CCM meta-model from GeCM

2) *ACM: an example of CCM refinement*

This level is a refinement of the DICM, where the specific characteristics and dependencies of the application domain are considered. Again, DSCM is based on the GeCM constructs, in other word DSCM is an instance of GeCM. Different DSCM would refine DICM for each of those target application domains. We choose to illustrate our example using ACM component model as a refinement of CCM component model for real time domain. As shown in Figure 5, The CCM component model is refined into different concepts as described in [22]:

ComponentTrigger. A number of internal methods within a component. Typically, they are used for record keeping and invariant checking of the component. These methods are further qualified by filling in attributes such as period, deadline and invariant.

ComponentHealthManager. It detects anomalies, identify and isolate the fault causes of those anomalies (if feasible), prognosticate future fault and mitigate effects of faults- on the level of individual components.

State Variable. They are similar to attributes in CCM but cannot be modified from outside component.

Parameters. They are configuration attributes which once set during initialization remain constant during the life cycle of that component

Method. All the ports - publisher, consumer, facet, and receptacle - have to finish their unit of work within a specified deadline. This deadline can be qualified as

HARD (strict) or SOFT (relatively lenient). A HARD deadline violation is an error that requires intervention from the underlying middleware. A SOFT deadline violation results in a warning. The soft deadline warning is sent to the component health manager. Other methods like Post-Condition define the contract to be satisfied at the end of the execution, Pre-condition, CallType and Read-only.

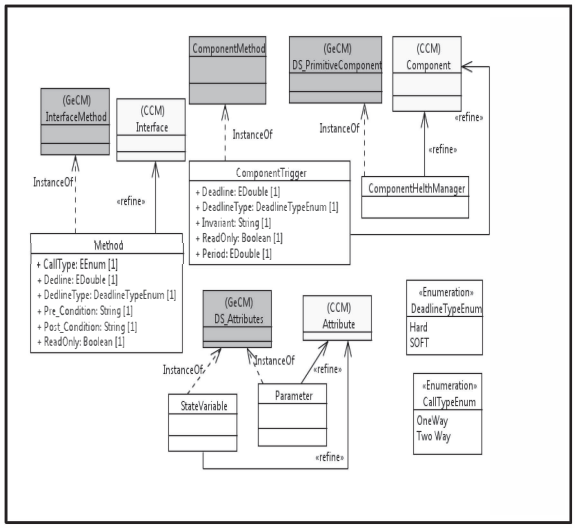

Fig. 5. ACM as an example of a DSCM from CCM

VI. CONCLUSION AND FUTURE WORK

In this paper, we have proposed an overview of concepts -of several component based approaches among academic and industrial fields - to build common representation of component models for several domains.

In this way this work gives several ways to go towards universal concepts to model components systems. Among these concepts, we can cite the followings: (1) high level of abstraction to capture all the facets of a component assembly, (2) The domain independent design and (3) The domain specific design.

We have proposed a novel approach to deal with domain concepts in architectural level of component application. We introduce domain concepts in model level by deriving two models (DICM and DSCM) from the proposed component metamodel. In other words, the proposed generic component metamodel (GeCM) reflects the domain independent and the domain specific levels. Domain specific elements must not be considered as an isolated aspect, but as an aspect present in all stages of a system development.

We are currently working on several domains like distributed and embedded domains to extend existing component models. We are interesting in Fractal refinement for real time and embedded systems. We are also developing tools supporting the proposed approach.

The next step is to propose a flexible component meta-model allowing both to capture and to specify interactions via models from different domains. We believe that the specification of interaction mechanisms using interaction pattern [29] are suitable to build more flexible and reusable software component system. Our goal is to embed these patterns in UML models in the form of profiles [30], i.e. we aim at applying interaction semantics into models through UML profiles using an MDE process (see Fig. 6). An initial work has been done in our team [31].

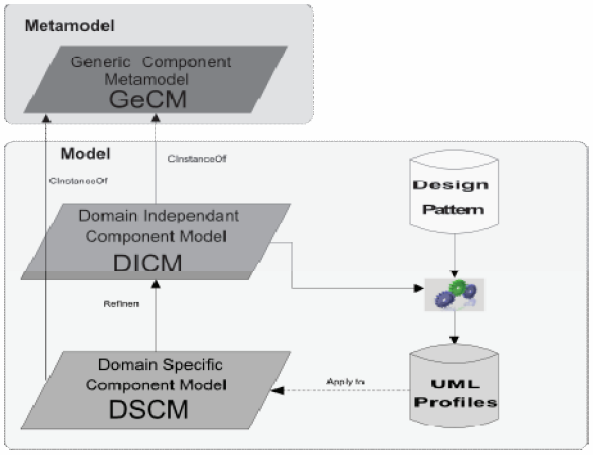

Fig. 6. Overview of Interaction Patterns integration

VII. REFERENCES

[1] Heinz W. Schmidt, Ivica Crnkovic, George T. Heineman, and Judith A.editors. Component-Based Software Engineering, 10th International Symposium, CBSE 2007, Medford, MA, USA, July 9-11, 2007, Proceedings, volume 4608 of Lecture Notes in Computer Science. Springer, 2007.

[2] A. W. Brown and K. C. Wallnau. The current state of CBSE. IEEE Software, 15(5):37-46, 1998.

[3] C. Szyperski. Component Software: Beyond Object-Oriented Programming. Addison-Wesley / ACM Press, Wesley, New York, 2002.

[4] D. Schmidt. Model-driven engineering. in IEEE computer, 39(2):41-47, 2006.

[5] Ivica Crnkovic, Séverine Sentilles, Aneta Vulgarakis, Michel R. V. Chaudron: A Classification Framework for Software Component Models. IEEE Trans. Software Eng. 37(5): 593-615 (2011)

[6] DeMichiel, L.G. (ed.): Enterprise JavaBeans Specification, Version 2.1. Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A, November 12 (2003)

[7] R. van Ommering, F. van der Linden, J. Kramer, and J. Magee. The Koala Component Model for Consumer Electronics Software. Computer, 33(3):78–85, 2000.

[8] Ananda Basu, Marius Bozga, and Joseph Sifakis. Modeling heterogeneous realtime components in BIP. In Proc. of the 4th IEEE International Conference on Software Engineering and Formal Methods, pages 3–12. IEEE, 2006.

[9] OMG, CORBA Component model, http://www.omg.org/technology/documents/formal/components.htm

[10] E. Bruneton, T. Coupaye, and J.B. Stefani. The fractal component model, 2004. Version 2.0-3.

[11] AUTOSAR Development Partnership, AUTOSAR – Technical Overview v2.0.1, 27/06/2006, Available at http://www.autosar.org/download/AUTOSAR_TechnicalOverview.pdf

[12] C. Szyperski, Component Software, Addison-Wesley Professional; 2002

[13] The Object Management Group, UML Superstructure Specification v2.1, April 2006, http://www.omg.org/docs/ptc/06-04-02.pdf

[14] G. Paul T. Francois J. Ackbar L. Kevin C. Geo□, B. Gordon, U. Jo, and S. Thirunavukkarasu. A generic component model for building systems software. ACM Trans. Comput. Syst., 26:1:1–1:42, March 2008.

[15] OMG. A uml profile for marte: Modeling and analysis of real-time embedded systems,beta 2. http://www.omgmarte.org/Documents/Specifications/08-06-09.pdf, June 2008.

[16] http://www.flex-eware.org/

[17] T. Bure˘s, P. Hn˘etynka, and F. Pl´a˘sil. SOFA 2.0 : Balancing Advanced Features in a Hierarchical Component Model. In Proc. of SERA, 2006.

[18] N.Diniz and J. Rufino. ARINC 653 in space. In data systems in Aerospace. European Space Agency, May 2005.

[19] Prochazka, M., Ward, R., Tuma, P., Hnetynka, P., Adamek, J.: A Component-Oriented Framework for Spacecraft On-Board Software. In: Proceedings of DASIA 2008, DAta Systems In Aerospace, Palma de Mallorca (May 2009)

[20] Lundbäck K-L., Lundbäck J., Lindberg M.: Component based development of dependable real-time applications Arcticus System.

[21] K. Balasubramanian, N. Wang, and D. C. Schmidt. Towards composable distributed real-time and embedded software, IEEE International Workshop on Object-Oriented Real-Time Dependable Systems, 0:226, 2003.

[22] Abhishek Dubey, Gabor Karsai, and Nagabhushan Mahadevan. 2011. A component model for hard real-time systems: CCM with ARINC-653. *Softw. Pract. Exper.* 41, 12 (November 2011), 1517-1550.

[23] H. Hansson, M. Akerholm, I. Crnkovic & M. Törngren, "SaveCCM: a component model for safety-critical real-time systems", EuroMicro Conference, Special Session Component Models for DependableSystems, Rennes, France, Sept. 2004.

[24] Yi Liu. 2003. Tutorial on the enterprise Javabeans (EJB) component model. J. Comput. Small Coll. 18, 6 (June 2003), 110-111.

[25] Object Management Group. Lightweight CORBA Component Model RFP, realtime/02-11-27 edition, Nov. 2002.

[26] William R. Otte, Aniruddha Gokhale, Douglas C. Schmidt, Johnny Willemsen, “Infrastructure for component-based DDS application development”, Proceedings of the 10th ACM international conference on Generative programming and component engineering, Pages: 53-62, October 2011.

[27] Object Management Group (OMG). OCL 2.0 Specification. http://www.omg.org/spec/OCL/2.2/, June 2005 Magee, J.,

[28] Dulay, N., Eisenbach, S., and Kramer, J. Specifying distributed software architectures. In Proceedings of the Fifth European Software Engineering Conference, ESEC'95, September 1995.

[29] C. Alexander. A Pattern Language: Towns, Buildings, Construction. Oxford University Press, USA., 1977.

[30] UML Superstructure 2.0 Draft Adpted Specification, http://www.omg.org/technology/ documents, 2004.

[31] Rahma Bouaziz, Brahim Hamid, and Nicolas Desnos, “Towards a better integration of patterns in secure component-based systems design”. In Proceedings of the 2011 international conference on Computational science and Its applications, (ICCSA'11) Vol. Part V. Springer-Verlag, Berlin, Heidelberg, page 607-621, 2011

[32] D. Box, Essential COM. Addison-Wesley, Reading, MA, 1997.

[33] Petr Hošek, Pop T., Malohlava M., Hnětynka P., Bureš T.: Supporting real-time features in a hierarchical component system, Tech. Report No. 2010/5, Dep. of Distributed and Dependable Systems, Charles University in Prague, December 2010.